Distributed Systems

Client-Server Models and Code Migration

WEEK 5: Client-Server Model, Code Migration, Mobile Code, Remote Execution, Distributed Application

Online Lecture Series - 5

Felix Edesa

Addis Ababa Science and Technology University

AASTU

Content Outline

Topics We Will Cover

- **Client-Server Model** — Request/response, real-world examples
- **Code Migration** — Motivation, strong vs. weak migration
- **Mobile Code** — Benefits, risks, use cases
- **Remote Execution** — Offloading computation, cloud services
- **Distributed Application** — Features, examples, scalability

AASTU

Client-Server Model

Key Points

- A distributed model where **clients request services** and **servers provide them**.
- Widely used in distributed systems and the Internet.
- Separates responsibilities: servers manage resources; clients consume them.

Example

- A web browser (client) requests a webpage from a web server.

AASTU

Client-Server Model: Industrial Examples

Real-World Applications

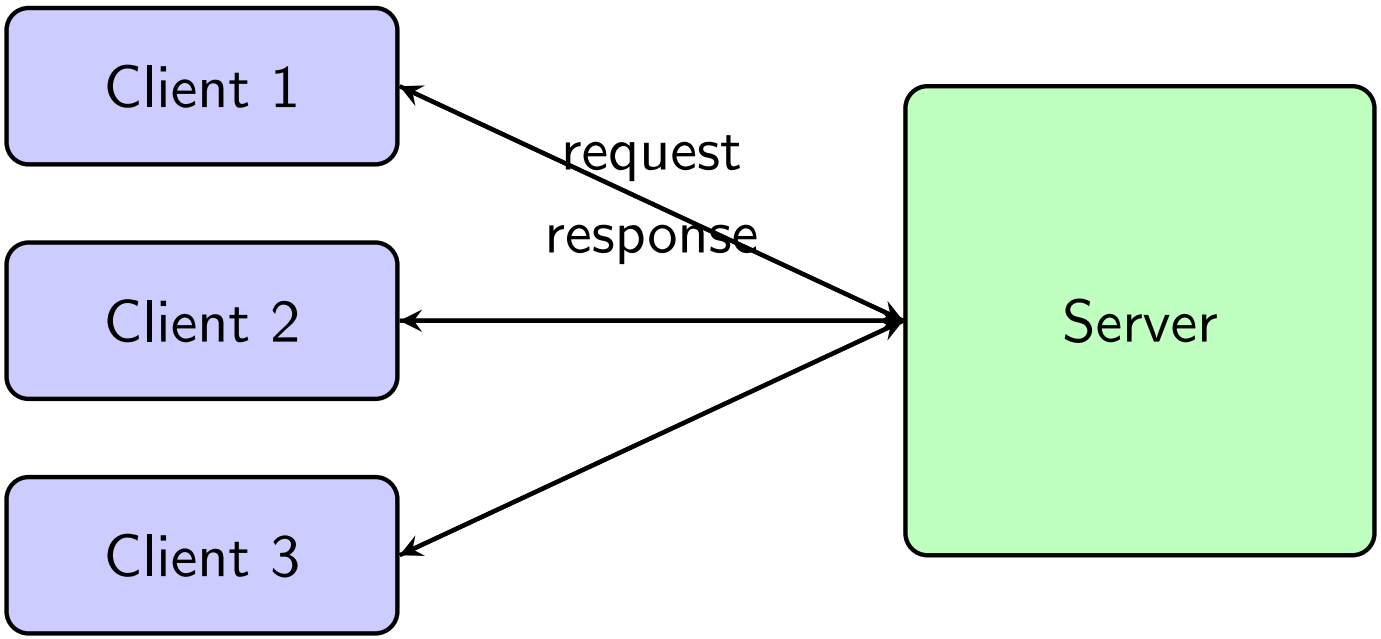
- **Banking Systems** — ATMs (clients) connect to central banking servers.
- **E-Commerce** — Clients (apps/websites) request product data from backend servers.
- **Cloud Services** — Clients use cloud APIs while servers handle computation/storage.

Observation

- The model is scalable but can face bottlenecks if the server is overloaded.

AASTU

Client-Server Model: Diagram



Advantages of Client-Server Model

Key Benefits

- **Centralized Control** – easier to manage security, backups, and updates.
- **Resource Sharing** – multiple clients can access the same data/services.
- **Scalability** – servers can be upgraded to handle more clients.

Example

- A university server hosting course materials for thousands of student clients.

AASTU

Limitations of Client-Server Model

Challenges

- **Single Point of Failure** – if the server crashes, all clients are affected.
- **Performance Bottleneck** – heavy load on server reduces responsiveness.
- **Maintenance Cost** – requires reliable server hardware and skilled admins.

Industrial Example

- Online banking downtime: a central server failure can block millions of transactions.

AASTU

Practical Example: Web Application

How It Works

- A user opens a browser (client) and requests `www.shop.com`.
- The request is sent to the server hosting the e-commerce website.
- The server processes the request and returns the product page.

Illustration

- **Client** = Web browser
- **Server** = Apache/Nginx hosting the online shop

AASTU

Client-Server in Industry

Examples

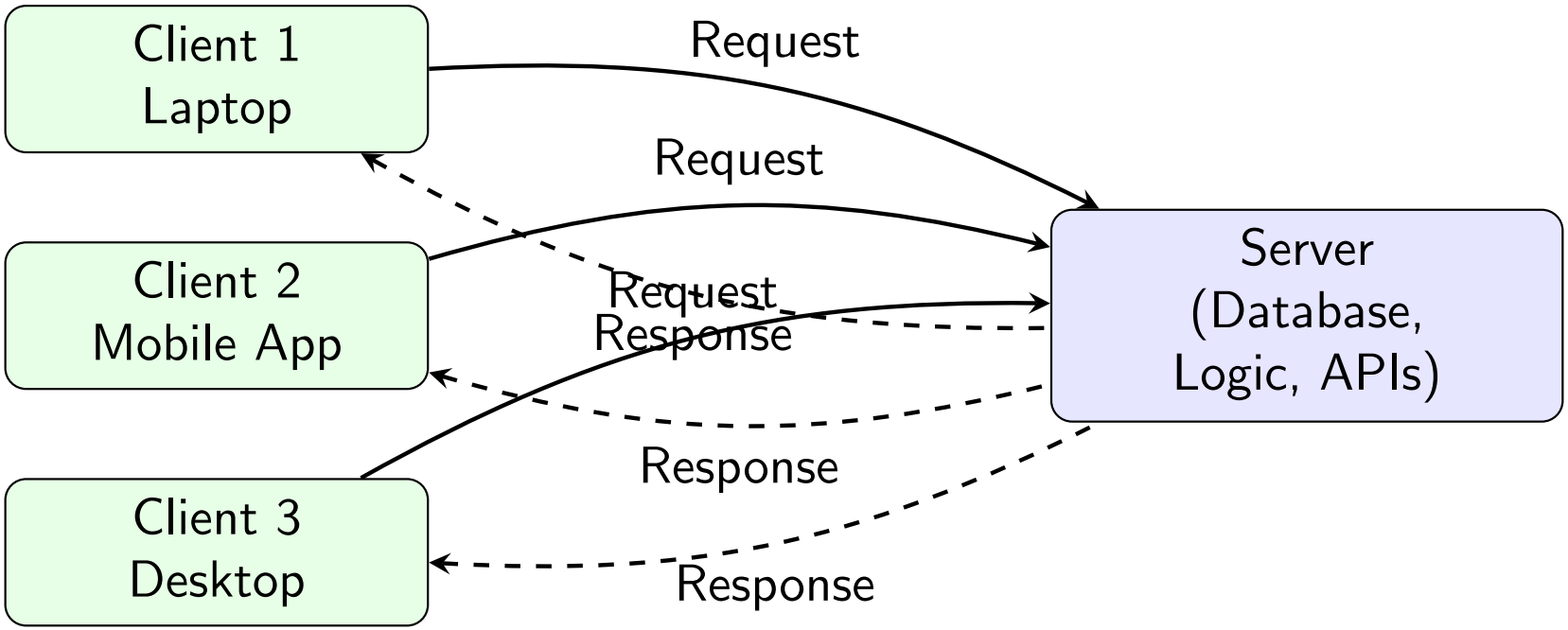
- **Healthcare** – doctors' terminals accessing patient databases on a central server.
- **Retail** – POS (Point of Sale) systems transmitting sales data to a central server.
- **Streaming Services** – Netflix servers deliver video content to client devices.

Observation

- Reliable, manageable, and scalable — ideal for mission-critical operations.

AASTU

Client-Server Architecture Diagram



Clients send requests and receive responses from the central server using a request-response model.

AASTU

Code Migration

Key Idea

- Code migration refers to moving programs or code segments from one machine to another during execution.
- Enables distributed systems to improve flexibility, load balancing, and performance.

Practical Example

- A web service dynamically moves a computation-heavy process to a more powerful remote server.

AASTU

Why Code Migration Matters

Motivations

- **Performance Optimization** – move computation closer to data.
- **Load Balancing** – distribute workload among multiple servers.
- **Fault Tolerance** – recover by migrating tasks from failed nodes.
- **Software Updates** – update or patch running systems dynamically.

Example

- Cloud-based video rendering tasks are moved between servers to minimize latency.

AASTU

Forms of Code Migration

Main Types

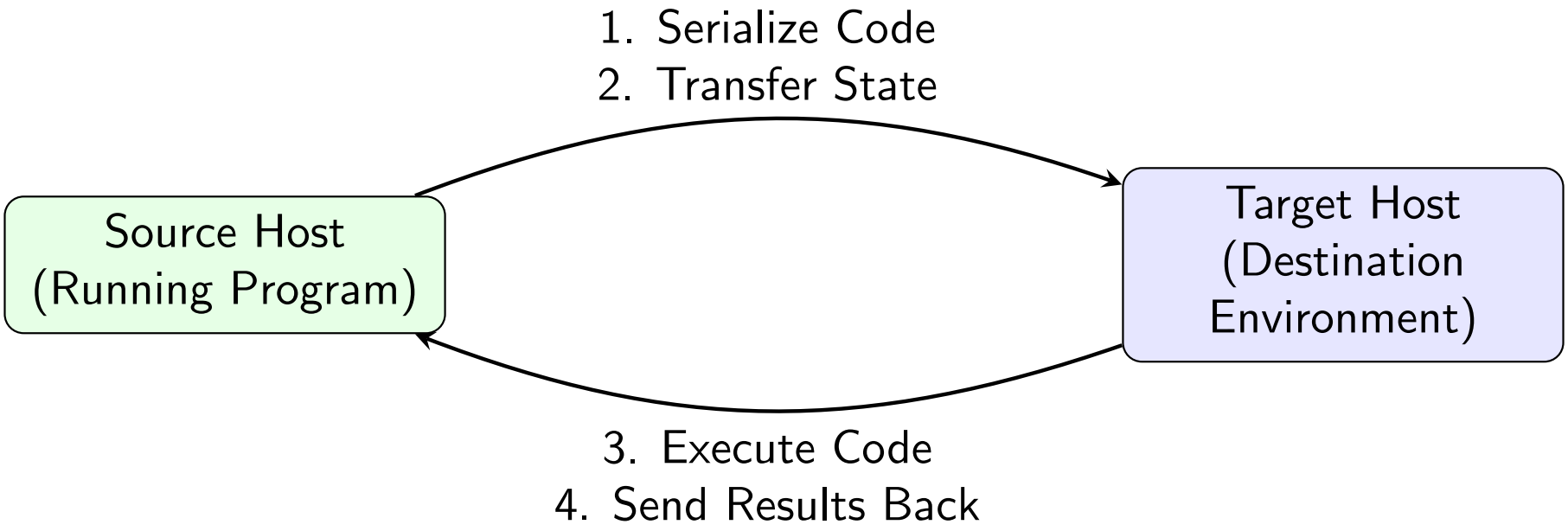
- **Weak Mobility** – only the code moves; execution restarts on the target machine.
- **Strong Mobility** – code and execution state move together and resume remotely.

Industrial Example

- Java Applets (weak mobility) – downloaded and executed on a client browser.
- Mobile Agents (strong mobility) – systems like IBM Aglets migrate active agents.

AASTU

Code Migration Process



Execution context
is transferred to a remote host
for processing and
results are sent back.

AASTU

Models of Code Migration

Execution Models

- **Remote Execution Model** – client sends computation to a powerful server.
- **Code-on-Demand Model** – server sends code to a client for local execution.
- **Mobile Agent Model** – code moves autonomously across hosts.

Example

- In web applications, JavaScript is downloaded (code-on-demand) and runs in the browser.

AASTU

Practical Example: Distributed Computation

Scenario

- A scientific computation platform offloads heavy tasks (e.g., simulation, rendering) to remote clusters.
- Each task migrates with its code and state, executes remotely, and sends results back.

Industrial Example

- SETI@home and Folding@home use distributed code migration for large-scale computation.

AASTU

Advantages and Challenges

Advantages

- Improved performance and resource utilization.
- Enhanced flexibility and scalability.
- Reduces data transfer by moving computation closer to data.

Challenges

- Security and trust issues in host environments.
- Handling heterogeneous architectures.
- Managing failures during migration.

AASTU

Summary of Code Migration

Essence

- Code migration makes distributed systems more adaptive and efficient.
- Commonly used in cloud computing, web systems, and mobile agents.
- Strong and weak mobility differ in how much state is transferred.

Looking Ahead

- Next: **Mobile Code** — how code travels autonomously and executes across heterogeneous systems.

AASTU

Optimal Migration Decision

Goal

- Determine whether migrating computation improves overall performance.
- Let T_l = local execution time, T_m = migration overhead, and T_r = remote execution time.

Decision Rule

Migrate if $T_m + T_r < T_l$

- T_m includes transfer, initialization, and re-synchronization costs.

AASTU

Python Demo: Code Migration Concept

Overview

- Demonstrates how computation can be migrated to another machine.
- Used in distributed systems for offloading tasks to remote nodes.
- Mimics how cloud functions or containers execute workloads remotely.

Goal

- Understand the process of sending code to a remote system for execution.
- Observe how results are returned to the local client.

AASTU

Code Migration: Setup Requirements

Environment Setup

- Python installed on both local and remote machines.
- Remote host must allow SSH connections.
- Install library: `pip install paramiko`.

Execution Flow

1. Connect securely to remote server via SSH.
2. Send code as a command to be executed.
3. Retrieve and display output locally.

AASTU

Code Migration via SSH: Shell Example

SSH Remote Execution

- Connect to a remote server:

```
ssh username@remote_host
```

- Send and execute a Python script remotely:

```
scp my_script.py username@remote_host:/tmp/
ssh username@remote_host python3 /tmp/my_script.py
```

- Optional: capture output locally

Python Example: Remote Execution (Part 1)

Define Remote Execution Function

```
import paramiko
def remote_execute(host, user, password, code):
    ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy
   (paramiko.AutoAddPolicy())
    ssh.connect(host, username=user,
   password=password)
    stdin, stdout, stderr = ssh.exec_command(code)
    out, err = stdout.read().decode(),
   stderr.read().decode()
    ssh.close(), return out, err
```

AASTU

Python Example: Remote Execution (Part 2)

Run and Retrieve Results

```
# Example: Migrate computation to remote host
code_to_run = print('Running remotely...')
out, err = remote_execute(
    '192.168.1.10', 'user', 'password',
    code_to_run
)
print("Output:", out)
if err:
    print("Error:", err)
```

Code is executed on a remote system — a simple simulation of code migration.

Mobile Code in Distributed Systems

Definition

- Mobile code refers to programs or scripts that **move across network nodes** and execute on remote machines.
- Unlike traditional code, it is **not bound to the originating host**.

Key Features

- **Portability:** Can run on different hosts with compatible runtime.
- **Autonomy:** Carries logic and instructions along with data.
- **Dynamic Deployment:** Can be sent and executed at runtime without pre-installation.

AASTU

Mobile Code: Benefits

Key Advantages

- **Flexibility** — Code can move to the location where resources or data reside.
- **Reduced Network Load** — Instead of transferring large datasets, send code to the data.
- **Dynamic Updates** — Systems can update behavior at runtime without restarting clients.

Example

- A monitoring script that moves to different servers to collect metrics instead of central polling.

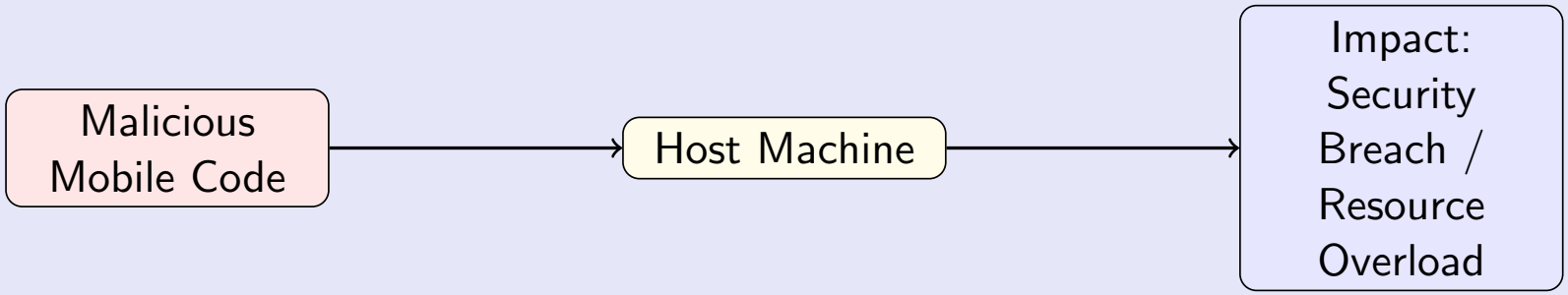
AASTU

Mobile Code: Risks

Challenges and Risks

- **Security Threats** — Malicious code could compromise host machines.
- **Compatibility Issues** — Different runtime environments may break code execution.
- **Resource Overuse** — Mobile code could consume CPU, memory.

Diagram: Risk Flow



AASTU

Mobile Code: Use Cases

Practical Applications

- **Distributed Monitoring** — Code moves to servers to collect logs and metrics locally.
- **Content Distribution** — Scripts for updating software or configurations on client machines.
- **IoT Devices** — Code is sent to sensors or embedded devices to update behavior remotely.

Industrial Example

- Smart home systems send small programs to devices to adjust behavior based on user preferences dynamically.

AASTU

Remote Execution in Distributed Systems

Definition

- Remote execution allows a program or task to **run on a different machine** than the one where it originated.
- Enables **load balancing, parallel processing, and resource optimization**.

Key Points

- Typically requires **network connectivity** and a compatible execution environment.
- Can be **synchronous** (waits for completion) or **asynchronous** (fires and forgets).

AASTU

Benefits of Remote Execution

Advantages

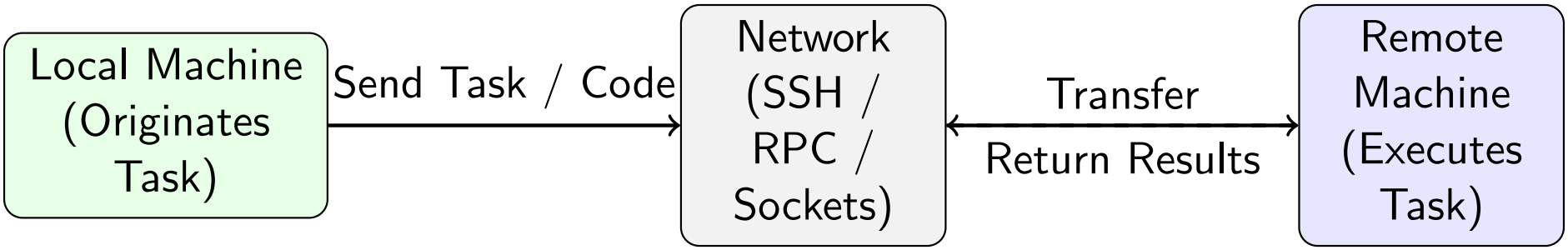
- **Resource Utilization:** Leverage idle machines for computation.
- **Scalability:** Run tasks on powerful remote servers.
- **Flexibility:** Deploy updates or patches without touching local systems.

Practical Example

- Running large data analysis on a remote cloud server instead of a local laptop.
- System administrators executing scripts on multiple servers via SSH.

AASTU

Remote Execution: Conceptual Diagram



Distributed Applications: Overview

Definition

- Applications where components run on multiple networked computers but appear as a single coherent system to users.
- Aim: improved performance, fault-tolerance, and scalability.

Key Features

- Concurrency: multiple processes run simultaneously.
- Transparency: location, replication, and access are hidden from the user.
- Scalability: system can grow without affecting functionality.

AASTU

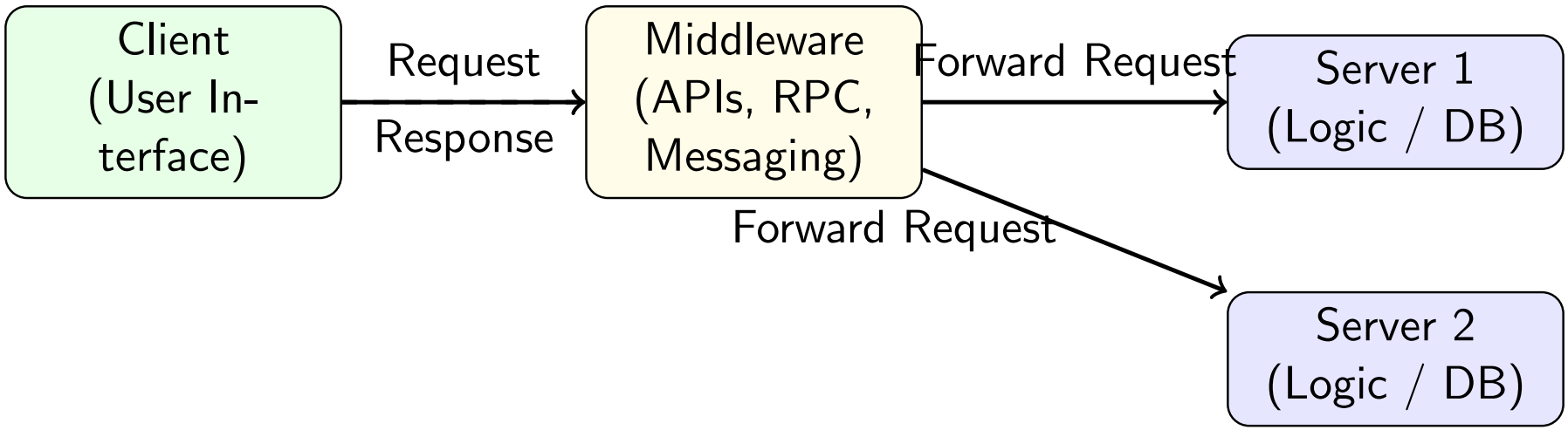
Distributed Applications: Examples

Industry Examples

- **Online Banking** — transactions processed across multiple servers for reliability.
- **E-commerce Platforms** — Amazon, eBay: multiple servers handle catalogs, orders, payments.
- **Streaming Services** — Netflix, YouTube: content delivery via distributed nodes for low latency.
- **Collaborative Tools** — Google Docs, Zoom: distributed processing for real-time collaboration.

AASTU

Distributed Application Architecture



Benefits of Distributed Applications

Key Advantages

- **Fault Tolerance** — if one node fails, others can continue processing.
- **Load Balancing** — distribute tasks to prevent bottlenecks.
- **Scalability** — new nodes can be added to handle increasing demand.
- **Resource Sharing** — multiple users share the same resources efficiently.

Example

- Content Delivery Networks (CDNs) like Akamai serve millions of users simultaneously with distributed servers.

AASTU

Risks and Challenges of Distributed Applications

Common Issues

- **Network Latency** — delays due to communication over distributed nodes.
- **Consistency** — ensuring all nodes have the same view of data.
- **Security** — data may traverse insecure networks
- **Debugging Complexity** — harder to track errors across multiple machines.

Industrial Example

- Banking system updates must be synchronized across multiple branches to prevent inconsistent account balances.

AASTU

Questions?

Thank you for your attention!

AASTU

References

Suggested References

1. M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., Version 4.02, February 2024.
2. G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5th ed., Addison-Wesley, 2012.
3. N. Lynch, *Distributed Algorithms*, 2nd ed., Morgan Kaufmann, 1996.
4. R. E. Bryant and D. O'Hallaron, *Computer Systems: A Programmer's Perspective*, 4th ed., Pearson, 2024.

AASTU